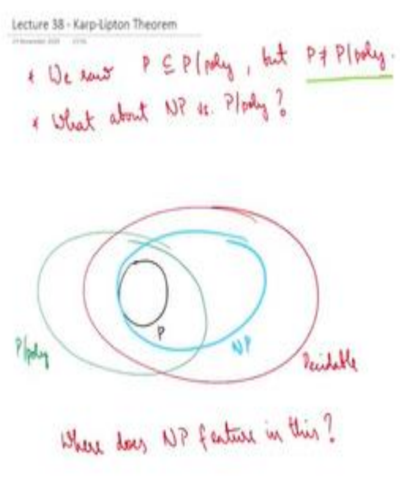
Computational Complexity

Prof. Subrahamanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Lecture - 38
Karp-Lipton Theorem

(Refer Slide Time: 00:15)



Lecture 38 - Karp-Lipton Theorem

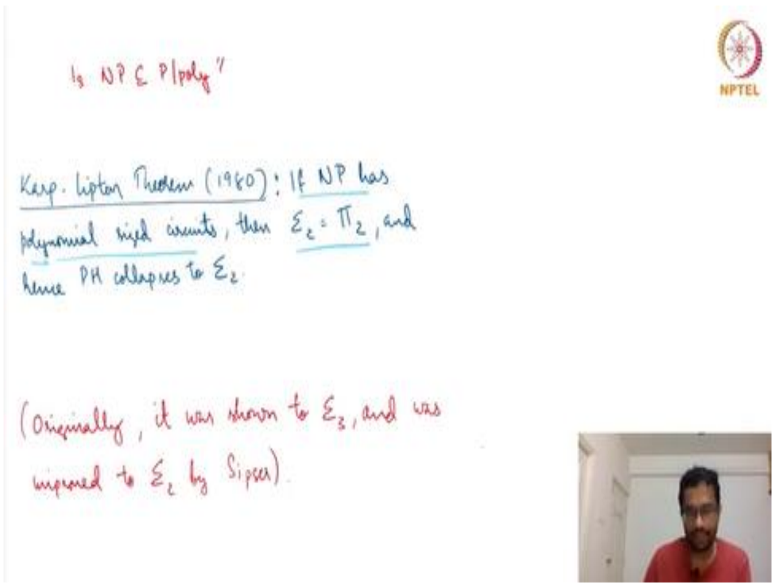
* We saw $P \subseteq P/poly$, but $P \neq P/poly$.
* What about NP vs. P/poly?

Where does NP feature in this?

Hello and welcome to lecture 38 of the course on computational complexity. So, in the last lecture we were discussing the computational complexity class P by poly which is a class of languages that can be decided by polynomial size circuits. We saw that P is contained in P by poly and we also saw that there are undecidable languages in P by poly hence P is not equal to P by poly. So, now that brought us to the question where does NP fit in all this?

Is NP a subset of P by poly or is it not really a subset? So, NP basically the question is NP like this contained in P by poly or is it like this, that there are languages in NP that are not in P by poly. Notice that NP cannot be a superset of P by poly because P by poly contains undecidable languages. So, in this lecture we will see how P by poly relates to other classes that we are already aware of starting with NP.

(Refer Slide Time: 01:38)



Is NP ⊆ P/poly?

Karp-Lipton Theorem (1980): If NP has polynomial sized circuits, then $\Sigma_2 = \Pi_2$, and hence PH collapses to Σ_2.

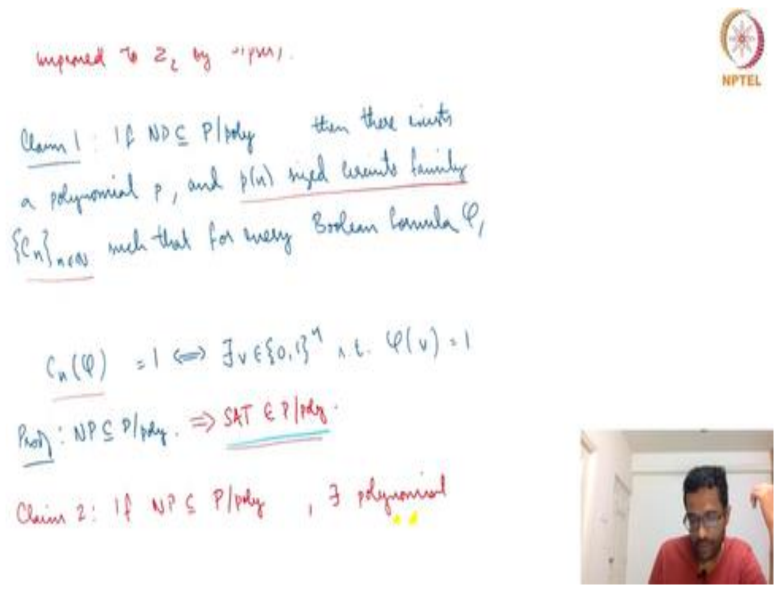
(Originally, it was shown to Σ_3, and was improved to Σ_2 by Sipser).

So, we start with the Karp Lipton theorem which provides some evidence to the fact that it is likely that NP is like this, NP is not a subset of P by poly. So, likely that NP is not a subset of P by poly, this is because if NP was a subset of P by poly, if NP had polynomial size circuits it would mean that the polynomial hierarchy collapsed. And it is believed that the polynomial hierarchy is strict and does not collapse.

Because of which we can view Karp Lipton theorem as evidence to the fact that NP does not have polynomial size circuits. In fact, it shows that the polynomial hierarchy collapses to the second level to sigma 2. So, Karp Lipton theorem was proved in 1980 by Richard Karp and Richard Lipton and the original proof had them showing that the polynomial hierarchy would collapse to sigma 3, the third level and which was later improved to sigma 2 by Zipser.

So, the proof might have slightly improved but the spirit of the theorem remains the same whether the polynomial hierarchy collapses to sigma 3 or sigma 2, both of them are somewhat equally unlikely, so this is what Karp Lipton theorem states.

(Refer Slide Time: 03:16)



improved to Σ_2 by $\cdot|p(n)|$.

Claim 1: If $NP \subseteq P/poly$ then there exists a polynomial p, and $p(n)$ sized circuits family $\{C_n\}_{n \in \mathbb{N}}$ such that for every Boolean formula φ,

$$C_n(\varphi) = 1 \iff \exists v \in \{0,1\}^n \text{ s.t. } \varphi(v) = 1$$

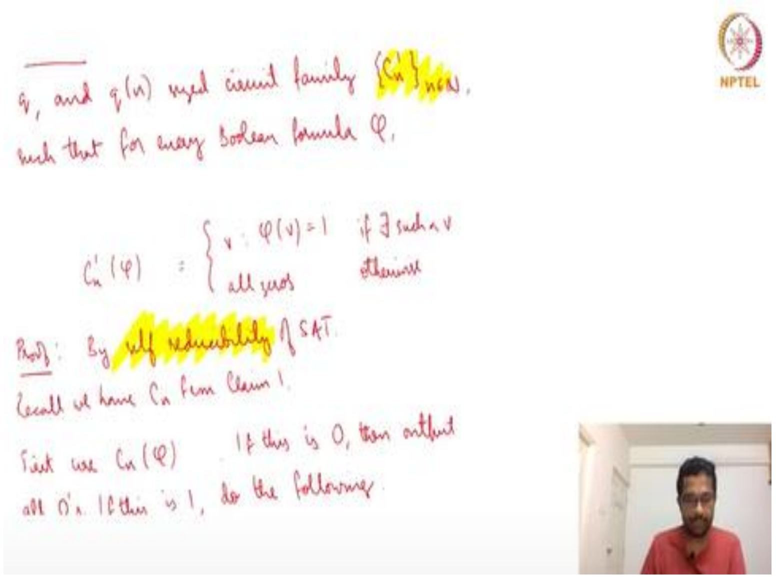
Proof: $NP \subseteq P/poly. \Rightarrow SAT \in P/poly.$

Claim 2: If $NP \subseteq P/poly$, $\exists$ polynomial

So, before getting into the proof of Karp Lipton theorem we will see an interesting result about the self-reducibility of SAT which will be made use of in the proof of Karp Lipton theorem. So, the first result is that if NP is contained in P by poly, then there are polynomial size circuits for SAT which means there are polynomial size circuit family that takes us and put a Boolean formula and outputs a 0, 1 output depending on whether the Boolean formula is satisfiable or not.

This is just more or less a restatement of the fact that SAT is contained in P by poly, so if NP is in P by poly obviously SAT is in P by poly and which means that there is a circuit family C n that gives us output 1 if and only if the Boolean formula is satisfiable. So, C n takes us and puts a Boolean formula and outputs 1 if and only if the Boolean formula phi is satisfiable which is what we have written here.

(Refer Slide Time: 04:31)



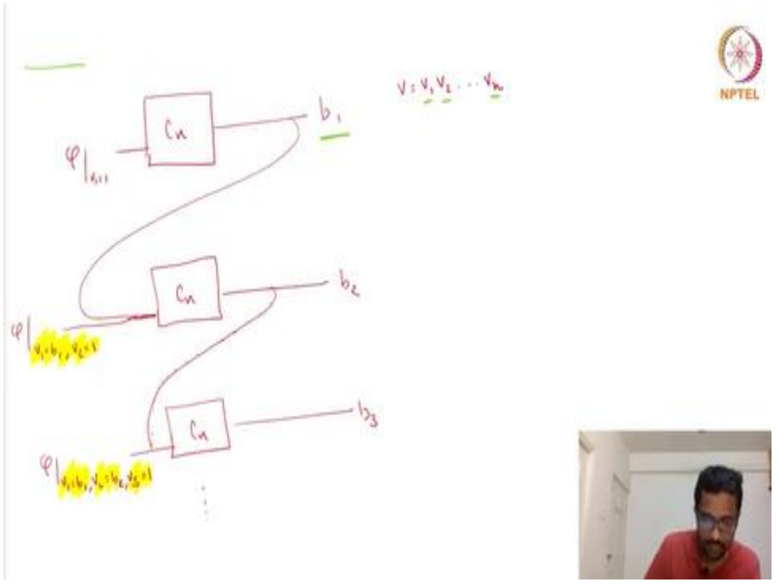
q, and $q(n)$ sized circuit family $\{C'_n\}_{n\in\mathbb{N}}$, such that for every Boolean formula φ,

$$C'_n(\varphi) = \begin{cases} v : \varphi(v) = 1 & \text{if } \exists \text{ such a } v \\ \text{all zeros} & \text{otherwise} \end{cases}$$

Proof: By self reducibility of SAT.
Recall we have C_n from Claim 1.
First use $C_n(\varphi)$. If this is 0, then output all 0's. If this is 1, do the following.

Claim 2 states that if NP is in P by poly, again the same assumption then there is another polynomial size circuit family called C n prime, the one I have highlighted here. That not only decides whether the formula is satisfiable or not, but also it gives a satisfying assignment for the formula if the formula is satisfiable. So, basically C n prime takes us input a Boolean formula and outputs a satisfying assignment if it exists, otherwise it just outputs all zeros.

(Refer Slide Time: 05:10)



And here is where we will make use of the self-reducibility of SAT, so what is self-reducibility? It is a property that is satisfied by SAT but also other NP languages but let us see what it is. So, what we do? The goal is to construct a satisfying assignment of given a Boolean formula phi. So, we first input the Boolean formula to the circuit family C n from the claim 1, if phi is satisfiable C n will output 1, if phi is not satisfiable C n will output 0.

Suppose C n outputs 0, this means that phi is not satisfiable and we can stop processing right now because the goal is to find a satisfying assignment. If the formula itself is not satisfiable, obviously there is no satisfying assignment that we can discover, so we can stop right there. So, if the output 0 then we can stop right there, if we output 1 then we will try to arrive at a satisfying assignment. So, if C n outputs 1 it means that the formula is satisfiable.

So, from now on whatever we say works only when the formula is satisfiable. So, now we can assume that the formula phi has a satisfying assignment, now what we do is we let the variables of the formula phi v 1 to v n. So, let the variables of the formula phi v 1 to v n, let these be the n variables of the formula. Now what we do is we fix the first variable to phi 1, we fix v 1 to be 1 and then feed in it into the circuit family C n or we feed it into circuit C n.

So, once we fix v 1 = 1, now we have a smaller formula because v 1 will be replaced by 1, peak 1 complement will be replaced by 0. So, some constants may lead to the formula becoming smaller and there are n - 1 variables. And now the circuit C n tests whether the resulting formula is satisfiable or not. Suppose the resulting formula is satisfiable which means there is a way to fix v 2, v 3 etcetera in such a way that v 1 being 1, there is a satisfying assignment.

Suppose the resulting formula is not satisfiable, this means that there is no satisfying assignment for phi where v 1 is equal to 1. So, let the output of this be b 1, so b 1 is 1 if and only if there is a satisfying assignment for phi with v 1 = 1 and b 1 = 0 if and only if there is a satisfying assignment for phi with v 1 = 0. So, if b 1 is 0, that means that there is no satisfying assignment where b 1 = 1 but obviously we started doing this only after ensuring that there is some satisfying assignment.

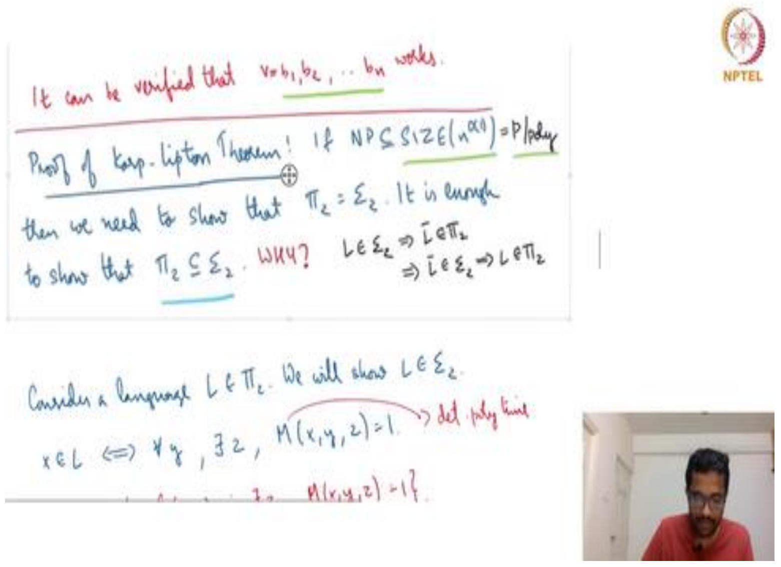
So, those satisfying assignments or assignments should have the first variable set to 0. So, what I am saying is that if b 1 = 1 or b 1 = 0 whatever be the value of b 1 there is a satisfying assignment for phi where v 1 is set to the same value as b 1. So, now what we can do is to do this again so now we know that there is a satisfying assignment where v 1 equals b 1, now we do this again. Now in the second stage we take b 1 and we set v 1 = b 1 and we again set v 2 = 1.

And then we feed that into C n. So, now again b 2 is what is output, so whether b 2 is 0 or 1 we know that there is a satisfying assignment for phi with v 1 set to b 1 and v 2 set to b 2. So, now knowing that again we could repeat the same thing and the third stage we input phi where v 1 is set to b 1, v 2 is set to b 2 and b 3 set to 1 and so on we can continue so we get b 3. At the end you can check that whatever we get as b 1, b 2 and b 3 and so on.

This n bit vector will be a satisfying assignment for the formula phi assuming that there is 1. So, this idea of constructing a satisfying assignment using just a black box that decides whether the formula is satisfiable or not is called self-reducibility of set. In fact, all the NP problems have the property of self-reducibility where from a given a black box for the decision version you could construct a certificate.

This in the case of SAT, the certificate is the satisfying assignment. Now this will be used crucially in the proof of the Karp Lipton theorem. Notice that what we do here? We just used the circuit family in claim 2 again and claim 1 again and again to generate a circuit family that outputs a satisfying assignment for the Boolean formula.

(Refer Slide Time: 11:04)



It can be verified that $v = b_1, b_2, \ldots b_n$ works.

Proof of Karp-Lipton Theorem: If $NP \subseteq SIZE(n^{O(1)}) = P/poly$ then we need to show that $\Pi_2 = \Sigma_2$. It is enough to show that $\Pi_2 \subseteq \Sigma_2$. WHY? $L \in \Sigma_2 \Rightarrow \bar{L} \in \Pi_2$
$\Rightarrow \bar{L} \in \Sigma_2 \Rightarrow L \in \Pi_2$

Consider a language $L \in \Pi_2$. We will show $L \in \Sigma_2$.

$x \in L \iff \forall y, \exists z, \overbrace{M(x,y,z) = 1}^{} \rightarrow$ det. poly time

$M(x,y,z) = 1\}$.

Now let us move to the proof of the Karp Lipton theorem. Again, this works assuming that NP has polynomial size circuits, so size of n to the power constant which is the same as P by poly. What do we do is? If NP has polynomial size circuits, we show that pi 2 is equal to sigma 2. It was earlier shown when we saw polynomial hierarchy that if pi 2 is shown to be equal to sigma 2 then the entire polynomial hierarchy will collapse to the sigma 2.

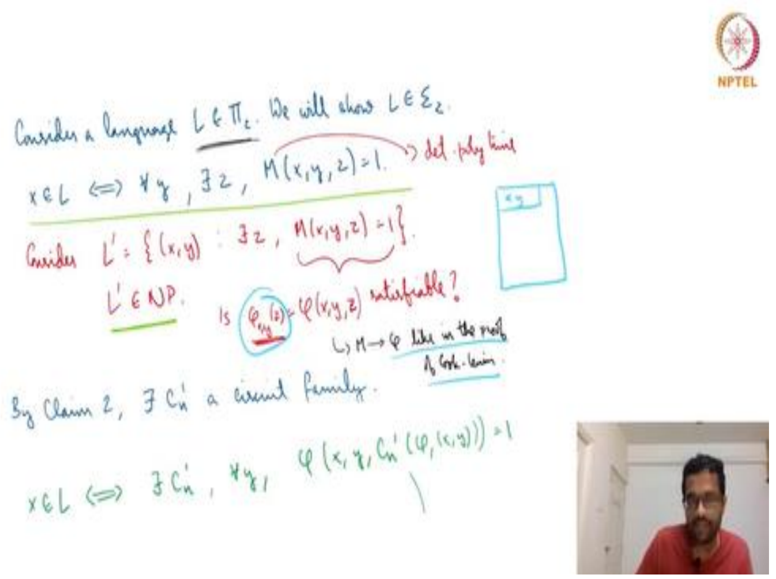
In fact, it is enough to show that pi 2 is contained in sigma 2, this is because pi 2 and sigma 2 are sort of complement classes L is in pi 2 if and only if L complement is in sigma 2. So, if we assume that pi 2 is in sigma 2, we can also show that sigma 2 is contained in pi 2, meaning if we assume that pi 2 is contained in sigma 2, we can also show that sigma 2 is containing pi 2 using the same argument, so the argument being this.

Suppose L is in sigma 2, so the goal is to show that sigma 2 is contained in pi 2. This means that L complement is in pi 2, because that is a definition of sigma 2 and pi 2. This means that again by assumption pi 2 is contained in sigma 2, L complement is in sigma 2 which means that L is in pi 2. So, we started with L in sigma 2 and then show that L is in pi 2. This is basically showing that sigma 2 is contained in pi 2.

So, the summary is that it is enough to show that pi 2 is contained in sigma 2, now that is what we will show. Assuming that NP has polynomial size circuits, in fact SAT has polynomial size circuits, we will show that pi 2 is contained in sigma 2. Again, the standard way to do this is to assume a general language or an arbitrary language in pi 2, let us call that L. Now it is in pi 2, so there is a standard pi 2 characterization that x is in L if and only if for all strings y of a certain polynomial length.

There exists a string z of another polynomial length such that a deterministic polynomial time machine M can verify that x, y, z or M can accept x, y, z. So, basically along with y and z, M accepts x. So, M is a deterministic polynomial time machine.

(Refer Slide Time: 13:55)



Consider a language $\underline{L \in \Pi_2}$. We will show $L \in \Sigma_2$.

$x \in L \iff \forall y, \exists z, M(x,y,z) = 1.$ $\rightarrow$ det. poly time

Consider $L' = \{(x,y) : \exists z, \underbrace{M(x,y,z) = 1}\}$.

$\underline{L' \in NP}$. Is $\boxed{\varphi_{x,y}(z)} = \varphi(x,y,z)$ satisfiable?

$\hookrightarrow M \rightarrow \varphi$ like in the proof of Cook-Levin.

By Claim 2, $\exists C_n'$ a circuit family.

$x \in L \iff \exists C_n', \forall y, \varphi(x, y, C_n'(\varphi, (x,y))) = 1$

So, this is a standard pi 2 characterization. We want to show that L is in sigma 2. So, we will try to obtain a sigma 2 characterization for L, so what we will try to do is to kind of exist an existential quantifier here. We will try to get rid of this there exists z, we will try to arrive at this or without this existential quantifier we will try to proceed, but at the end we will see that we need an existential quantifier before the for all y, so this will move over here before the for all y.

So, it will not be there exists z, it will be there exists something else but when we move it from here to here it will end up being a sigma 2 characterization, so this is what we will try to do. So, first let us notice a related language, a language related to L. So, consider L prime, L prime consists of all the pairs x, y such that there exists a z which leads to M accepting x, y, z. So, it is up to all the pairs x, y for which there is a z that leads to M accepting x, y, z.

So, this is clearly in the NP form relation, it is in the NP form because x, y can be thought of as just one string and this x, y is accepted if and only if there is a proof or witness z, that leads to it being verified. So, L prime is in NP, so now L prime is an NP, so you could think of an NP machine for L prime. So, equivalently you could think of something like this and you could think of the proof of Cook Levin theorem.

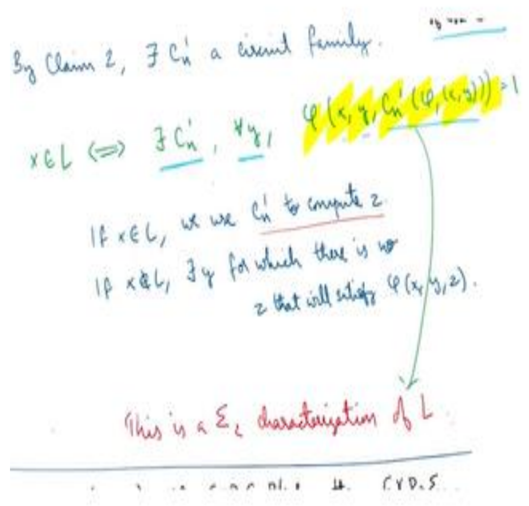
So, in this case x and y are in input and it needs to find z, x and y are fixed as part of the machine and you need to find z that leads to M accepting it. So, just like in the proof of Cook Levin theorem where we came up with the Boolean formula that is satisfiable if and only if M accepts a string x, just like that we can find a Boolean formula or a circuit formulation for this deterministic polynomial time machine M x, y, z such that, so let us call it phi of x, y, z.

In fact, once x and y are fixed, we could write it like this phi of xy, phi xy of z meaning x and y are fixed so we do not need to make it an argument to the formula, so z is only an argument. So, now the question is, is the phi of xy satisfiable? So, that is the question here. Once again, the construction of phi from M is just like in the proof of Cook Levin theorem where we given a nondeterministic Turing machine.

We are trying to write an equivalent Boolean formula which will be satisfiable if and only if nondeterministic Turing machine accepts a certain input. So, now we have this Boolean formula phi of x, y which takes as input is z. In claim 2 we saw a way or we saw a circuit family that actually outputs a satisfying assignment. So, recall what I said above that we want to get rid of this for all there exists z here, this existential quantifier.

So, now what we will do is z is the satisfying assignment for this Boolean formula phi x, y and we will use the circuit family constructed in claim 2 to generate the z, so the claim 2 said that if SAT has polynomial size circuits, there is a circuit family that gives you the satisfying assignment. So, that satisfying assignment is the z and we will use a circuit family to generate that z. So, the goal is to generate that z and remove the z part.

(Refer Slide Time: 18:36)


By Claim 2, $\exists C_n'$ a circuit family.

$x \in L \iff \exists C_n', \forall y, \varphi(x, y, C_n'(\varphi,(x,y))) = 1$

If $x \in L$, we use C_n' to compute z

If $x \notin L$, $\exists y$ for which there is no z that will satisfy $\varphi(x, y, z)$.

This is a Σ_2 characterization of L

So, we will use the circuit family to generate that z but we do not know the circuit family. Claim 2 just says that there exists a circuit family if SAT is in P by poly, so it does not matter whether we know what it is or not, we do not need to know it, we could just use non-determinism to guess the circuit family. So, we use an existential quantifier again to guess the circuit family, so we say that there exists a circuit family C n prime such that for all y.

Now because we assume the existence of C n prime, we could use C n prime itself to generate the z. So, basically to phi we give as input x and y which is already fixed and then we use C n prime to generate z and if it is a satisfiable x and y then C n prime would generate a correct z and if it is not a satisfiable x and y, C n prime would not generate a correct z. So, let us see why this is a proper characterization? Suppose x was in L meaning that for all y there is a z such that Mx, M x, y, M x, y, z is equal to 1.

That means C n prime, there is a z that satisfies Mx, M x, y, M x, y, z and C n prime can be used to generate that z. So, C n prime will be generating that set and there is such a C n prime because of the assumption and claim 2. And we use C n prime to compute that z, so it will lead to x being accepted. Suppose x is not in L, then that means this thing is not correct for all y there is a z such that it gets accepted is not correct.

So, which means the negation should be correct that means there exists a y such that for all z, M x, y, z = 0. So, there is a y such that for all z M x, y, z = 0 meaning there is no z that will satisfy the phi x, y, z. So, whatever the circuit that we guess, whatever the circuit outputs it will not be able to result in the satisfiability or success it will not result in a satisfying assignment for phi. So, if x is not in L, then this phi will not be satisfied.

This equation here will not satisfy this Boolean formula. So, that means that this characterization is a legitimate proper characterization. It satisfies both cases when x is in L, it satisfies when x is not in L, it again does not satisfy and we can see that there axis for all characterization and also phi is a polynomial sized formula because it was generated like in the proof of Cook Levin theorem. And C n prime is also a polynomial size formula as per the proof of claim 2.

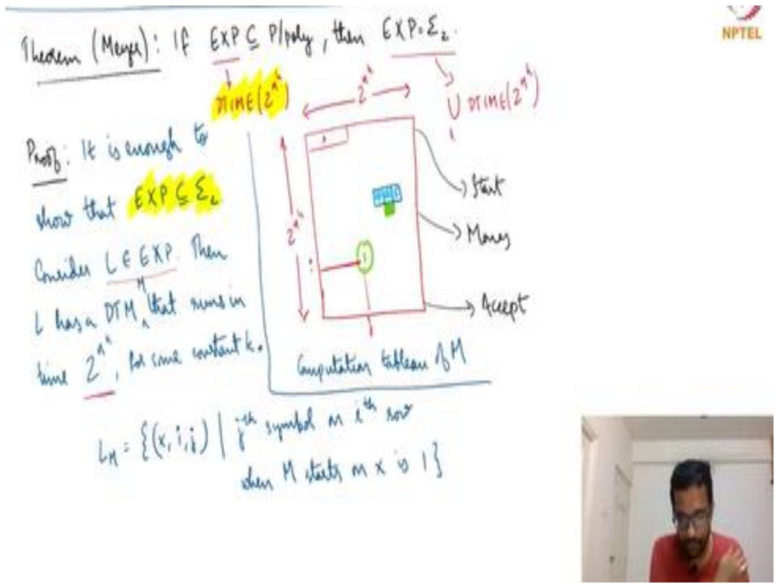
So, both of this entire thing is polynomial size, it is a polynomial time verifier. Hence it is a proper sigma 2 characterization of the language L and that is all we wanted to show. Basically, what we did is given this first pi 2 characterization of L, we arrived at a sigma 2 characterization, meaning we took an arbitrary language from pi 2 and then showed that it is in sigma 2. All that we did is to use self-reducibility to get rid of this there exists a z quantifier.

We use self-reducibility to assume the existence of a circuit family that can generate the z. So, instead of saying there exists a z we assume the existence of the circuit family, there exist a circuit family which is able to compute the z and then everything else was automatic, that is a proof of Karp Lipton theorem. So, again it states that if NP has polynomial size circuits, then polynomial hierarchy will collapse to sigma 2.

So, again this is used to kind of convince people that it is very unlikely that NP has polynomial size circuits. In complexity theory there are other results of this type as well, where we say that unfortunately we are unable to make unconditional statements, we are not able to say that NP, NP does not have polynomial size circuits for sure. So, what we do is things like this, if NP had polynomial size circuits.

Then something else happens which is something very very unlikely, this is the kind of statement that we have. Possibly this is the first such statement if unlikely event a or if a happens then b happens and we know that b is even more unlikely or b is unlikely, so hence a is also an unlikely kind of inference.

(Refer Slide Time: 24:20)



Theorem (Meyer): If EXP $\subseteq$ P/poly, then EXP $=\Sigma_2$.

Proof: It is enough to show that EXP $\subseteq \Sigma_2$

Consider $L \in$ EXP. Then L has a DTM M that runs in time 2^{n^k}, for some constant k.

$L_M = \{(x, i, j) \mid j^{th}$ symbol in i^{th} row when M starts in x is $1\}$

So, the next theorem that I want to state is Meyer's theorem. So, this is also very similar to the Karp Lipton theorem in spirit, so let us see what that states. This states that if exponential time has polynomial size circuits, then exponential time is contained in sigma 2 or exponential time is equal to sigma 2. So, what is exponential time? Exponential time is actually a union of all k D time, 2 power n power k.

So, it is a 2 power polynomial, so there is a polynomial in the exponent of 2 and we will show that exponential time is contained in sigma 2. Obviously, the other direction is easy because sigma 2 is contained in polynomial P space which is an x prime, it is something that you can easily verify. So, we will focus on showing that exponential time is contained in sigma 2 assuming that it has polynomial size circuits.

So, again with the standard thing we assume an arbitrary language in x time, we call it L and since L is an exponential time, L has a deterministic Turing machine M that runs in exponential time, so that runs in 2 power n power k for some constant k. So, DTM m that runs in 2 power n power k. Now let us consider the computational table of M, the one on the right side. And this computational table or computational tableau has 2 power n power k rows and 2 power n power k columns.

Because it runs the number of columns or the width of this computational table indicates how much the tape can be used. So, in 2 power n power k time it can be used up to 2 power n power k cells in the tape in the 2 power n power k cells in the tape could be used. And the height of the table is the actual time that is taken by the Turing machine which is also 2 power n power k. So, it takes some x as input and then at the end makes some computation.

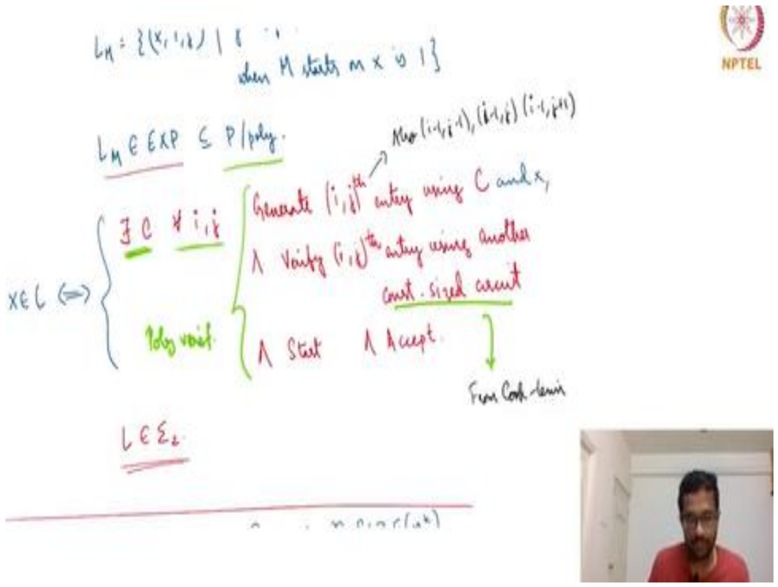
And decides to accept and reject something. So, both heights as well as width are both exponential. Now let us consider an auxiliary language L M, what is L M? L M is the set of all triplets x, i, j which says that the ith symbol, so maybe I will just change it a bit, I think the jth symbol on the ith row is 1 when M starts on the input x. So, this is the ith row and the jth symbol on that row which is the column j and this entry is written to be 1 when M starts on the input x.

So, L M consists of all the triplets x, i, j such that if M starts on input x, the i, jth entry is 1 in the computational table. So, if the computation table uses other symbols, let us say the tape alphabet consists of other symbols you could always consider a binary representation of the tape and binary representation of the table and consider the same similar kind of language. Obviously, L M is in x time, this language is an x time.

Because you could one way to decide this language is to actually run the Turing machine M on the input x and run up to i time steps and reach the ith row and see what is the jth cell of the computational table. So, obviously it runs the overall running time is exponential time but here we are only running up to the ith row so even that is exponential time, so hence L M is obviously in exponential time.

And by assumption we assume that exponential time has polynomial size circuits, by assumption L M is in P by poly meaning; given x, i and j are given M x, i and j, there is a polynomial size circuit family that can decide whether when M runs on this input x is the i, jth entry 1 or 0. So, now this is all that is enough to construct a sigma 2 characterization for the language L, so recall that we started with the language L in exp.

(Refer Slide Time: 30:11)



$L_M = \{(x, 1^i, 1^j) \mid$ [illegible] when M starts on x is $1\}$

$L_M \in EXP \subseteq P/poly$.

$x \in L \iff \exists C \ \forall i, j$ Generate $(i,j)^{th}$ entry using C and x, $\wedge$ Verify $(i,j)^{th}$ entry using another const. sized circuit $\wedge$ Start $\wedge$ Accept.

Also $(i-1, j-1), (i-1, j), (i-1, j+1)$

Poly veri.

From Cook-Levin

$L \in \Sigma_2$

Now we will derive a sigma 2 characterization for L. So, by assumption L is in P by poly which means there is a circuit family C that decides L, so there is a circuit family C that decides L. But we do not know what C is, C could be anything, C could be some trade circuit family. But we do not need to know what C is because we could just guess it. So, we use the existential quantifier to guess the existence of C but it may give something which will verify that this case is correct.

Now what do we need to check, in order to check that x is in L ideally, we want to verify the entire computation in this computation table, we want to verify that the start is proper, the start is proper then all the moves are proper and finally the string gets accepted. So, moves meaning it each level we need to check that at the configuration at the i plus first row is this valid successor to the configuration in the ith row and these needs to be done for all the i.

So, the first row is a proper starting configuration and then the second row is a valid successor of first row, third rows the valid successor of second row and so on and then finally we need to accept. So, all this needs to be checked. Unfortunately, this seems like an exponentially big table, so there are exponentially many things to check. But what we will do is, we will make use of the non-determinism to ensure and we will cleverly use for all quantifier to do all the checks at once.

So, let us see how that happens. The main thing here is that like we have said before, the computation in a Turing machine is an extremely local phenomenon. So, consider these three blue cells; this one, this one and this one and consider the green cell just beneath it in the next row. So, the entry in the green cell is only determined by the three blue cells just above it, because the computation in the Turing machine is an extremely local phenomenon.

And this is for this green cell but this is the same thing is true for any cells. So, maybe somewhere else there is some other set of other sets of cells and they also behave similarly, so some other set of cells somewhere else is similar, this black cell is generated by the entries in the cells above it. And this computation can be done by a constant sized formula or a circuit family, but then this constant size circuit family needs to be repeated exponentially many times to make a thorough check of the all the entries in this table.

And we do not have in sigma 2 even though we have quantifiers finally we should only have a polynomial sized or polynomial a verifier that runs in polynomial time. So, we will use the quantifiers to take care of exponential things, so that the verifier only does polynomial verification, so let us see how that happens. We need to check the start; we need to check the moves and we need to check the accept.

So, the start is a polynomial size verification, we just need to check the first row is something. Accept is also polynomial verification, we just need to check whether there is a q accept. What really have the potential to kind of kill us is the moves there are exponentially many cells to check. So, what we do is? We have this for all quantifiers where we check for all cells i j what we do is, we generate the I, jth entry using C and x.

So, C generates the i, jth entry and we already assumed that x time is in P by poly and L is in x prime, so there is a circuit that will tell us whether the i, jth entry is 1 or not. So, generate the i, jth entry using C and x and then you can also generate the entries just above it, the i - 1 j - 1, i - 1 j and i - 1 and j + 1. All these you can generate so maybe also i - 1 j - 1, i - 1 j, i + 1 j + 1. So, all this is generated.

And then we verify using the constant size circuit which is what we saw in the proof of Cook Levin theorem that the i, jth entry is a proper successor of the previous rows, this is the constant size circuit, so this is obtained from the proof of Cook Levin theorem. We saw the same idea; we are using it for probably the third or fourth time now that computation is local and each cell can be computed by a constant size computation.

And then as I said checking the; start configuration, the accepting configuration which is not really that much. So, the key thing here is the existence of the circuit C is something we just assumed and we needed to have a proper, we use the existential quantifier to guess the circuit C. And then we use for all quantifier to automate the checking. So, for all will run through all the possible i and j and we do not need to check because what remains is just checking the i, jth entry.

Because checking all the entries is taken care of by the quantifier. So, this again becomes polynomial. The verification after for all there exists this becomes a polynomial, this is a polynomial verification, so that is what is being done here. So, again we may be guessing a lot of junk circuits but it does not matter if there is a correct formula or if there is a correct sequence of configurations then there is a circuit family that guesses it.

And one of the guesses will end up guessing the correct circuit family and the correct entries ijth entries for all the rows and columns and that will lead to the successful verification and acceptance of the input string x. So, that is, we started with L being an x time language and now we are giving a polynomial, we are showing that it is existing in the polynomial hierarchy in the second level, so sigma 2 language.

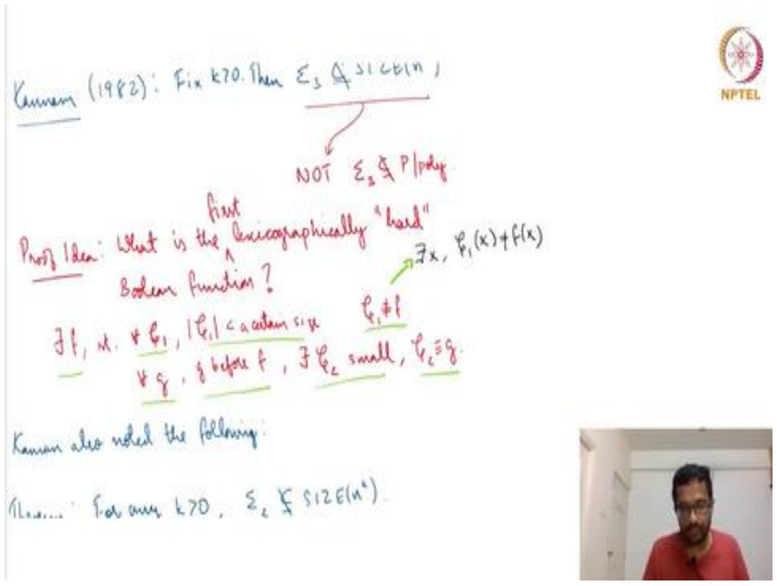
So, we first guess the circuit and then for all i and j, so notice that i and j are also polynomial sized, the range of i and j are exponential but the description of i and j, to write down i and j we require only log of 2 power n power k which is n power k bits. So, all of this is polynomial size and the resulting verification is also polynomial size, so that shows that L is in sigma 2. So, we are showing that an arbitrary language in exponential time by smartly using the quantifiers to guess the circuit.

And then using the quantifiers to run through all the i and j for all quantifiers, we are able to reduce it to a polynomial verifier. So, if exponential hierarchy had polynomial size circuits, then it would mean that exponential hierarchy not exponential hierarchy, exponential time complexity class exponential time at polynomial size circuits then that would mean that this exponential time is actually contained in sigma 2.

So, which would also imply that the polynomial hierarchy collapses because the entire polynomial hierarchy is a subset of exponential time so if all of that collapses to sigma 2, even P space collapses to sigma 2. Again, this is also another theorem of the form if x happens, then y happens. And we know that y is extremely unlikely, so x is also extremely unlikely. So, this is the correct way to interpret this, is to say that it is unlikely for x prime to have polynomial size circuits.

So, one may wonder where is x prime in this picture? So, it is maybe between certainly decidable but it is above NP, so it may be somewhere in between. But it cannot be a subset of P by poly, in fact it actually says that it collapses to sigma 2.

(Refer Slide Time: 39:53)



Kannan (1982): Fix $k>0$. Then $\Sigma_3 \not\subseteq SIZE(n^k)$

NOT $\Sigma_3 \not\subseteq P/poly$

Proof Idea: What is the first lexicographically "hard" Boolean function?

$\exists f$, st. $\forall C_1$, $|C_1| <$ a certain size $C_1 \neq f$ → $\exists x$, $C_1(x) \neq f(x)$

$\forall g$, g before f, $\exists C_2$ small, $C_2 \equiv g$.

Kannan also noted the following:

Thm....: For any $k>0$, $\Sigma_2 \not\subseteq SIZE(n^k)$.

The third theorem that I want to talk about today is Kannan's theorem which says that if you fix a k, it picks a particular exponent, let us say k is 10 then it says that the polynomial hierarchy contains languages that requires circuits or circuit family of size bigger than n power 10. So, you fix a k, so fix k to be 10 then it says that polynomial hierarchy contains languages that require circuits for which a circuit family will require more than n power 10 size and this holds for any k.

So, if you see the statement in a textbook, you may think that this says that sigma 3 is not contained in the size of n power k and this is not the same as saying that sigma 3 is not contained in P by poly. This is not the same as saying sigma 3 is not contained in P by poly. What it is saying is that if you give k, give this particular exponent like 10 then I will give you a language that cannot be decided by a circuit family of size n power 10.

If you tell me 20 then I will give another language which cannot be decided by a circuit family of size n power 20. So, in other words there are languages in the polynomial hierarchy that require arbitrarily large polynomial size circuits but again this is not a proof that sigma 3 is not a subset of P by poly simply because the language is different for each exponent and the language that it gives you to show that L is not in size of n power 10, may be fitting in n power 11.

So, it is still in P by poly. But there is yet another language that will not be in size of n power 11, so what Kannan showed is that sigma 3 is not contained in size of n power k for any fixed k meaning he showed a specific language that is not in n power k. So, I will not show the proof but I will just tell you the proof idea, it is a very smart application of the quantifiers. So, all that he does is he is asking what is the first lexicographically hard Boolean function.

So, suppose you can write down all the functions in a certain order like maybe in a dictionary, in a dictionary there is a cryptographic ordering. So, what is the first hard function? So, hard meaning a function that requires more than a certain size, so we saw in Shannon's theorem that there are functions that require some exponential size, 2 power n divided by 10n, is not enough to capture certain functions. So, let us consider one such function.

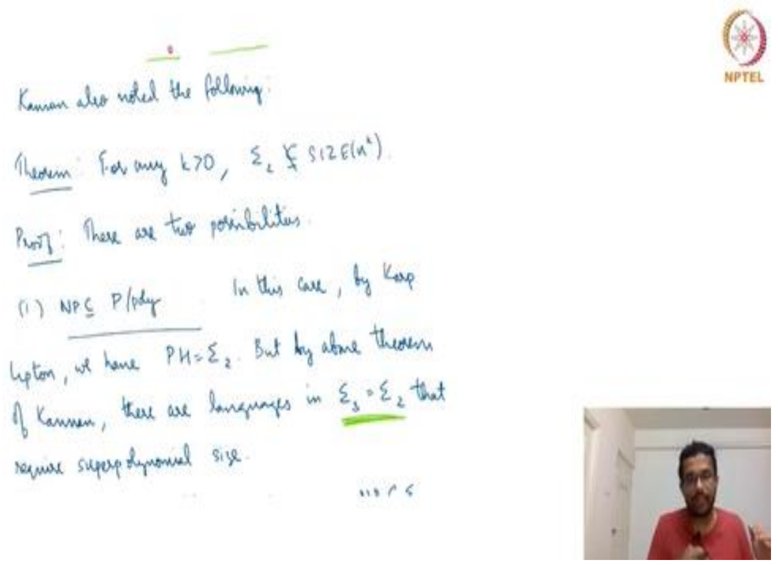
What is the first function that requires 2 power n divided by 10 in size. Now how do we write this question using quantifiers, so one way to write it is something I have scribbled here let us see. There exists f, so we want to first check that f cannot be decided by small sized circuits, meaning for all circuit families C 1, where C 1 is of a small size, C 1 is not the same as f and we also want to ensure that it is the first hard function.

So, the way to ensure that for all g or for all functions g where g appears lexicographically before f, g is equivalent to a small circuit family. So, there is a small circuit family C 2 such that C 2 is equal to g, so we already have used there exists f such that for all C 1 something, for all g something, so here there exists C 2 something. But now you can notice that how do we check that C 1 is equal to g or C 2 is equal to g or C 1 is not equal to f.

We may need yet another quantifier, so C 1 not equal to f requires us to say there exists a specific input, let us say x such that C 1 of x is not the same as f of x. So, this we will write it like that exists f such that C 1 of x is not equal to f of x and similarly if we want to say C 2 = g, we will have to say for all y, C 2 of y = C g of y. So, you can see why some more quantifiers are required, so there are two parallel sets of quantifiers some of which can be combined.

And this leads to a sigma 3 characterization or perhaps sigma 4 characterization. So, Kannan showed sigma 3 but perhaps what I am writing is sigma 4, you can check that. So, all that the proof boils down to representing this what is the first lexicographically hard Boolean function, hard meaning something that requires more than 2 power n divided by 10n size.

(Refer Slide Time: 46:04)



Kannan also noted the following.

Theorem: For any $k > 0$, $\Sigma_2 \nsubseteq SIZE(n^k)$.

Proof: There are two possibilities.

(1) $NP \subseteq P/poly$. In this case, by Karp Lipton, we have $PH = \Sigma_2$. But by above theorem of Kannan, there are languages in $\Sigma_3 = \Sigma_2$ that require superpolynomial size.

So, it shows that in sigma 3 there are functions that require size arbitrarily in R size. Now this can be further improved using the Karp Lipton theorem, so we saw that the Karp Lipton theorem states that if SAT has polynomial size circuits, then polynomial hierarchy collapses to sigma 2. Now we will use the Karp Lipton theorem to improve the sigma 3. So, this sigma 3 we will improve it to sigma 2.

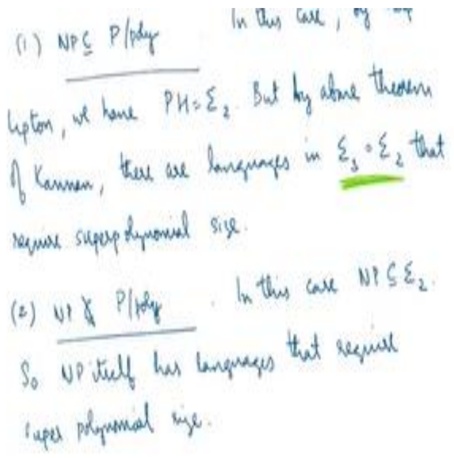
So, we will use that to show that there are even at the second level, even at the sigma 2 level there are languages in sigma 2 that require arbitrarily large polynomials, that much size. So, actually we do not give a again even in the case of lexicographically hard Boolean function, the proof is not really constructive. We just use the quantifiers to establish the existence of such a function. In this case it is even more indirect; there is one more level of indirection involved here.

So, there are two cases, again it is not constructive, there are two cases; first is that NP is contained in P by poly, so maybe it is better to write P by poly and P is contained in P by poly. So, now by Karp Lipton, we have that the entire polynomial hierarchy collapses to sigma 2. And we have already kind of seen the idea that sigma 3 contains languages that require arbitrarily large polynomial size.

But if NP is contained in P by poly, then sigma 3 everything in the polynomial hierarchy collapses to sigma 2, so in particular sigma 3 is equal to sigma 2. So, there are languages in sigma 2 itself that require arbitrarily large size, so which is what the statement is, so we have shown the statement if NP has languages or if all of NP has polynomial size circuits, then polynomial hierarchy collapses to sigma 2, so sigma 2 itself has languages of that require arbitrary large polynomials.

Because sigma 3 also coincides with sigma 2 and we already saw that sigma 3 has its languages. The second possibility is that NP is not contained in P by poly; this is even easier actually.

(Refer Slide Time: 49:01)



(1) $\underline{NP \subseteq P/poly}$ In this case, by [illegible] Lipton, we have $PH = \Sigma_2$. But by above theorem of Kannan, there are languages in $\Sigma_3 = \Sigma_2$ that require superpolynomial size.

(2) $\underline{NP \not\subseteq P/poly}$. In this case $NP \subseteq \Sigma_2$.

So NP itself has languages that require super polynomial size.

So, we wanted to show that there are hard functions in polynomial hierarchy and we are trying to show that it is in sigma 2. But if NP is itself not in P by poly, that means there are languages in NP itself that require super polynomial size circuits, super polynomial means more than polynomial size circuits. So, there are languages in NP itself that require more than polynomial size circuits and well NP is contained in sigma 2.

So, if we show that there are languages in NP that require super polynomial size circuits, that is enough to show that the language is in sigma 2. So, in either case whether NP is in P by poly or not in P by poly, there is a language in sigma 2 or even below that, which requires circuits of arbitrary large polynomial size. So, again this is just to give different theorems and how P by poly relates to the polynomial hierarchy and exponential time and so on.

So, we saw three theorems in this lecture; one is the Karp Lipton theorem, which states that if NP has polynomial size circuits, then polynomial hierarchy collapses to sigma 2 and two is Meyer's theorem, which says that if x time, exponential time had polynomial size circuits then exponential time collapses to sigma 2, the statement looks very similar to Karp Lipton theorem and third, the Kannan's theorem which says that, you tell me a specific polynomial.

And sigma 3 contains languages that require bigger than that polynomial size. So, if you tell me n power 10, then sigma 3 itself has languages that require more than power 10 size circuits. And this is improved to sigma 2 by using an application of Karp Lipton. So, this also Kannan himself observed in the same paper and I think Kannan's theorem was in 1982 and that is all I wanted to say in this particular lecture and thank you.